611301

Load Balancing Strategies for Multi-Block Overset Grid Applications

M. JAHED DJOMEHRI

Computer Sciences Corporation, NASA Ames Research Center, Moffett Field, CA 94035

RUPAK BISWAS

NAS Division, NASA Ames Research Center, Moffett Field, CA 94035

NOE LOPEZ-BENITEZ

Department of Computer Science, Texas Tech University, Lubbock, TX 79409

Abstract

The multi-block overset grid method is a powerful technique for high-fidelity computational fluid dynamics (CFD) simulations about complex aerospace configurations. The solution process uses a grid system that discretizes the problem domain by using separately generated but overlapping structured grids that periodically update and exchange boundary information through interpolation. For efficient high performance computations of large-scale realistic applications using this methodology, the individual grids must be properly partitioned among the parallel processors. Overall performance, therefore, largely depends on the quality of load balancing. In this paper, we present three different load balancing strategies for overset grids and analyze their effects on the parallel efficiency of a Navier-Stokes CFD application running on an SGI Origin2000 machine.

1 Introduction

The multi-block overset grid technique [2] is a powerful method for high-fidelity computational fluid dynamics (CFD) simulations about complex aerospace configurations. The solution process uses a grid system that discretizes the problem domain by using separately generated but overlapping structured grids that periodically update and exchange boundary information through Chimera interpolation [12]. However, to reduce time-to-solution, high performance computations of large-scale realistic applications using this overset grid methodology must be performed efficiently on state-of-the-art parallel supercomputers. Fortunately, a message passing paradigm can be readily employed to exploit coarse-grained parallelism at the grid level as well as communicate boundary data between distributed overlapping grids.

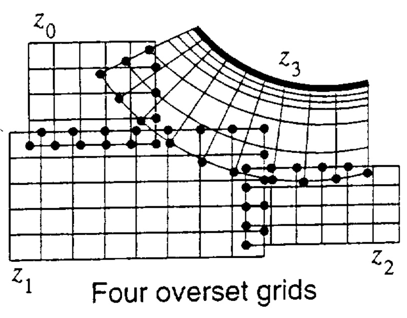
The parallel efficiency of the overset approach, however, depends upon the proper distribution of the computational workload and the communication overhead among the processors. Applications with tens of millions of grid points may consist of many overlapping grids. A smart partitioning of the individual grids (also known as blocks or zones) among the processors should therefore not only consider the total number of grid points, but also the size and connectivity of the inter-grid data. In fact, to be more accurate, the computational and communication times ought to be used to perform the load balancing. Major challenges during the clustering process may arise due to the wide variation in block sizes and the disparity in the number of inter-grid boundary points.

In this paper, we present three different load balancing strategies for overset grid applications. The first uses a simple bin-packing algorithm to maintain uniform computational loads across processors while retaining some degree of connectivity among the grids assigned to each processor. The other two load balancing techniques are more sophisticated and based on graph partitioning. One uses the spectral bisection algorithm available within the Chaco package [5] to balance processor workloads and minimize total communication. The other uses a task assignment heuristic from EVAH [8] to minimize the total execution time. We analyze the effects of all three methods on the parallel efficiency of a Navier-Stokes CFD application called OVERFLOW-D [9]. Our experiments are conducted on an SGI Origin2000 machine at NASA Ames Research Center using a test case that simulates complex rotorcraft vortex dynamics. The grid system consists of 857 blocks and approximately 69 million grid points. Results indicate that graph partitioning-based strategies perform better than a naive bin-packing algorithm and that the EVAH task assignment heuristic is generally superior to the Chaco spectral technique.

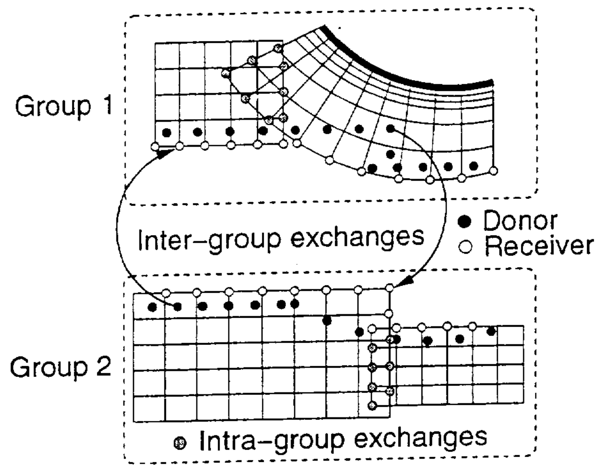
The remainder of this paper is organized as follows. Section 2 provides a brief description of the OVERFLOW-D overset grid application. The three load balancing techniques are described in Section 3. Parallel performance results are presented and critically analyzed in Section 4. Finally, Section 5 concludes the paper with a summary and some key observations.

2 Overset Grid Application

In this section, we provide an overview of the overset grid CFD application called OVERFLOW-D, including the basics of its solution process, the Chimera interpolation of inter-grid boundary data,



(a)



(b)

Figure 1: (a) Example of an overset grid system; (b) schematic of intra-group and inter-group communication.

and a message-passing parallelization model.

2.1 Solution Process

The high-fidelity overset grid application called OVERFLOW-D [9] is a special version of OVERFLOW [2] that owes its popularity within the aerodynamics community due to its ability to handle complex configurations. These designs typically consist of multiple geometric components, where individual body-fitted grids can be constructed easily about each component. The grids are either attached to the aerodynamics configuration (near-body) or detached (off-body). The union of all near- and off-body grids covers the entire computational domain. An overset system consisting of four grids (z_0, z_1, z_2, and z_3) is shown in Fig. 1(a).

Both OVERFLOW and OVERFLOW-D use a Reynolds-averaged Navier-Stokes solver, augmented with a number of turbulence models. However, unlike OVERFLOW which is primarily meant for static grid systems, OVERFLOW-D is explicitly designed to simplify the modeling of components in relative motion (dynamic grid systems). For example, in typical rotary-wing problems, the near-field is modeled with one or more grids around the moving rotor blades. The code then automatically generates Cartesian background (wake) grids that encompass these curvilinear near-body grids. At each time step, the flow equations are solved independently on each zone in a sequential manner. Overlapping boundary inter-grid data is updated from previous solutions prior to the start of the current time step using a Chimera interpolation procedure [12]. The code uses finite differences in space, with a variety of spatial differencing and implicit/explicit temporal time-stepping.

2.2 Chimera Interpolation

The Chimera interpolation procedure [12] determines the proper connectivity of the individual grids. Adjacent grids are expected to have at least a one-cell (single fringe) overlap to ensure the continuity of the solutions; for higher-order accuracy and to retain certain physical features in the solution, a double fringe overlap is sometimes used [13]. A program named Domain Connectivity Function (DCF) [10] computes the inter-grid donor points that have to be supplied to other grids (see Fig. 1). The DCF procedure is incorporated into the OVERFLOW-D code and fully coupled with the flow solver. All boundary exchanges are conducted at the beginning of every time step based on the interpolatory updates from the previous time step. In addition, for dynamic grid systems, DCF has to be invoked at every time step to create new holes and inter-grid boundary data.

2.3 Message Passing Parallelization

A parallel message passing (based on MPI) version of the OVERFLOW-D application has been developed around its multi-block feature which offers a natural coarse-grained parallelism [16]. The top-level computational logic of the sequential code consists of a time-loop and a nested grid-loop. Within the grid-loop, solutions are obtained on the individual grids with imposed boundary conditions, where the Chimera interpolation procedure successively updates inter-grid boundaries after computing the numerical solution on each grid. Upon completion of the grid-loop, the solution is advanced to the next time step by the time-loop. The overall procedure may be thought of as a Gauss-Seidel iteration.

To facilitate parallel execution, a grouping strategy is required to assign each grid to an MPI process. The total number of groups, G, is equal to the total number of MPI processes, P. Since a grid can only belong in one group, the total number of grids, Z, must be at least equal to P. If Z is larger than P, a group will consist of more than one grid. However, the parallel efficiency of the overset approach depends critically on how the grouping is performed. In other words, the individual grids must be partitioned among the processes so that the computational workload is balanced and the communication overhead is minimized. Three different techniques for clustering grids into groups in a load balanced fashion are discussed in Section 3.

In the MPI version of OVERFLOW-D, the grid-loop is therefore split into two nested loops: one over the groups (called the group-loop) and the other over the grids within each group. Since each MPI process is assigned to only one group, the group-loop is performed in parallel, with each process performing its own sequential grid-loop. The inter-grid/intra-group boundary updates among the grids within each group are conducted as in the serial case. However, inter-group exchanges between the corresponding processes are performed via MPI calls using send/receive buffers where donor points from grids in one group contribute to the solution at receiver points in another group (see Fig. 1(b)). The communication can be synchronous or asynchronous, but the choice significantly affects the MPI programming model.

The current version of OVERFLOW-D uses asynchronous message passing that relaxes the communication schedule in order to hide latency. Unlike the original synchronous model [16], these non-blocking invocations place no constraints on each other in terms of completion. Receive completes immediately, even if no messages are available, and hence allows maximal concurrency. In general, however, control flow and debugging can become a serious problem if, for instance, the order of messages needs to be preserved. Fortunately, in the overset grid application, the Chimera boundary updates take place at the completion of each time step, and the computations are independent of the order in which messages are sent or received. Being able to exploit this fact allows us to easily use asynchronous communication within OVERFLOW-D.

3 Load Balancing Strategies

Proper load balancing is critically important for efficient parallel computing. The objective is to distribute equal computational workloads among the processors while minimizing the inter-processor communication cost. On a given platform, the primary procedure that affects the load balancing of an overset grid application is the grid grouping strategy. To facilitate parallel execution, each grid must be assigned to an MPI process. In other words, the Z grids need to be clustered into G groups, where G is equal to the total number of processes, P. Unfortunately, the sizes of the Z grids may vary substantially and there may be wide disparity in the number of inter-grid boundary points. Both these factors complicate the grouping procedure and affect the quality of overall load balance.

In this section, we present three different grouping strategies for overset grid applications. The first uses a simple bin-packing algorithm while the other two are more sophisticated techniques based on graph partitioning. All three methods depend only on the characteristics of the grids and their connectivity; they do not take into account the topology of the physical processors. The assignment of groups to processors is somewhat random, and is taken care of by the operating system usually based on a first-touch strategy at the time of the run.

3.1 Bin-Packing Algorithm

The original parallel version of OVERFLOW-D uses a grid grouping strategy based on a bin-packing algorithm [16]. It is one of the simplest clustering techniques that strives to maintain a uniform number of "weighted" grid points per group while retaining some degree of connectivity among the grids within each group. Prior to the grouping procedure, each grid is weighted depending on the physics of the solution sought. The goal is to ensure that each weighted grid point requires the same amount of computational work. For instance, the execution time per point belonging to near-body grids requiring viscous solutions is higher than that for the inviscid solutions of off-body grids. The weight can also be deduced from the presence or absence of a turbulence model. The bin-packing algorithm then sorts the grids by size in descending order, and assigns a grid to every empty group. Therefore, at this point, the G largest grids are each in a group by themselves. The remaining $Z - G$ grids are then handled one at a time: each is assigned to the smallest group that satisfies the connectivity test with other grids in that group. The connectivity test only inspects for an overlap between a pair of grids, regardless of the size of the boundary data or their connectivity to other neighboring grids. The process terminates when all grids are assigned to groups.

3.2 Chaco Graph Partitioning Heuristic

Graph partitioning has been used in many areas of scientific computing for the purpose of load balancing. For example, it is frequently utilized to order sparse matrices for direct solutions, decompose large parallel computing applications, and optimize VLSI circuit designs. It is particularly popular for load balancing parallel CFD calculations on unstructured grids [15] and handling dynamic solution-adaptive irregular meshes [1].

As an analogy to this unstructured-grid approach, we can construct a graph representation of the OVERFLOW-D overset grid system. The nodes of the graph correspond to the near- and off-body grids, while an edge exists between two nodes if the corresponding grids overlap. The nodes are weighted by the weighted number of grid points and the edges by the communication volume. Given such a graph with Z nodes and E edges, the problem is to divide the nodes into G sets such that the sum of the nodal weights in each set are almost equal while the sum of the cut edges' weights is minimized. Such a grouping strategy is, in theory, more optimal than bin-packing in that the grids in each group enjoy higher intra-group dependencies with fewer inter-group exchanges.

Two popular graph partitioning software packages have been interfaced with OVERFLOW-D. The first is METIS [7], a widely used multilevel partitioner that first reduces the size of the graph by collapsing edges, then partitions the coarse graph, and finally projects it back to the original. Unfortunately, METIS is primarily suitable for large graphs and does not perform well with overset grid systems since the graph representations usually contain at most a few hundred nodes.

The second package, called Chaco [5], includes a variety of heuristic partitioning algorithms based on inertial, spectral, and Kernighan-Lin (KL) multilevel principles, as well as a few other simpler strategies. All algorithms employ global partitioning methods, but KL is a local refinement technique that can be coupled to global methods to improve partition quality. We have experimented with several Chaco partitioners, but the results reported in this paper were all obtained using the spectral algorithm. Spectral methods use the eigenvectors of the Laplacian matrix of the graph to partition it. The spectral algorithm in Chaco is a weighted version of recursive spectral bisection (RSB) [11] and uses a Lanczos-based eigensolver. A detailed description of the algorithm can be found in [6].

3.3 EVAH Task Assignment Heuristic

The other graph partitioning-based algorithm that we have implemented in OVERFLOW-D to address the grid grouping problem uses the EVAH package [8]. EVAH consists of a set of allocation heuristics that considers the constraints inherent in multi-block CFD problems. It was initially developed to predict the performance scalability of overset grid applications on large numbers of processors, particularly within the context of distributed grid computing across multiple resources [3]. In this work, we have modified EVAH to cluster overset grids into groups while taking into account their relative overlaps. The overall goal is to minimize the total execution time.

Among several heuristics that are available within EVAH, we have used the one called *Largest Task First with Minimum Finish Time and Available Communication Costs* (LTF_MFT_ACC). In the context of the current work, a task is synonymous with a block in the overset grid system. The size of a task is defined as the computation time for the corresponding block. LTF_MFT_ACC is constructed from the basic *Largest Task First* (LTF) policy that sorts the tasks in descending order by size. LTF is then enhanced by the systematic integration of the status of the proces-

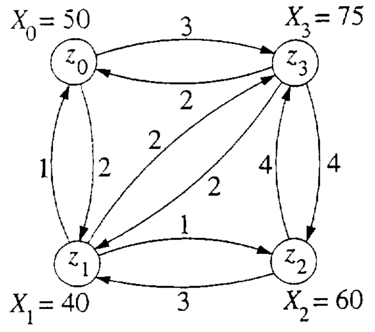
```
begin procedure LTF_MFT_ACC
1:    sort tasks $z_i$, $i = 1, 2, \ldots Z$ in descending order by size (LTF policy)
2:    for each processor $p_j$, $j = 1, 2, \ldots P$ do $\mathcal{T}(p_j) = 0$
3:    for each sorted task $z_i$, $i = 1, 2, \ldots Z$ do
3.1:      assign task $z_i$ to processor $p_j$ with minimum $\mathcal{T}(p_j)$ (LTF_MFT policy)
          $\mathcal{T}(p_j) = \mathcal{T}(p_j) + X_i$
3.2:      for each task $z_r \in R(z_i)$ assigned to processor $p_k \neq p_j$ do $\mathcal{T}(p_j) = \mathcal{T}(p_j) + C_{ir}$
3.3:      for each task $z_d \in D(z_i)$ assigned to processor $p_m \neq p_j$ do $\mathcal{T}(p_m) = \mathcal{T}(p_m) + C_{di}$
4:    end for
end procedure
```

Figure 2: Outline of the LTF_MFT_ACC task assignment heuristic from EVAH.

sors in the form of *Minimum Finish Time* (LTF_MFT). Finally, because of the overhead involved due to data exchanges between neighboring zones, LTF_MFT is further augmented to obtain the LTF_MFT_ACC task assignment heuristic. To model their impact on the overall execution time, LTF_MFT_ACC utilizes communication costs which are estimated from the inter-grid data volume and the interprocessor communication rate. A procedure has been developed to interface the DCF subroutine of OVERFLOW-D with EVAH heuristics. An outline of the LTF_MFT_ACC procedure is presented in Fig. 2.

It is easiest to explain the LTF_MFT_ACC grouping strategy by using a simple example and then stepping through the procedure in Fig. 2. Figure 3(a) shows a graph representation of the overset grid system in Fig. 1(a) that is being partitioned across two processors. The computational time for block z_i is denoted as X_i and shown for all four blocks in Fig. 3(a). Similarly, the communication overhead from block z_d (donor) to another block z_r (receiver) is denoted as C_{dr} and shown for all inter-grid data exchanges along the graph edges. In step 1, the four blocks are sorted in descending order by computational time; hence the order is: z_3, z_2, z_0, z_1. In step 2, the total execution times of the two processors are initialized: $\mathcal{T}(p_0) = \mathcal{T}(p_1) = 0$. Step 3 has to be executed four times since we have four grids that must be grouped. Block z_3 is assigned to processor p_0 and $\mathcal{T}(p_0) = 75$ in step 3.1. Since no other blocks have yet been assigned, steps 3.2 and 3.3 are not executed.

Now z_2 must be mapped to a processor. According to the LTF_MFT policy, z_2 is assigned to the processor that currently has the smallest total execution time; thus, z_2 goes to p_1 and $\mathcal{T}(p_1) = 60$



1:	sorted z_3, z_2, z_0, z_1	3.1:	$z_0 \to p_1$
2:	$\mathcal{T}(p_0) = \mathcal{T}(p_1) = 0$		$\mathcal{T}(p_1) = 64 + 50 = 114$
3.1:	$z_3 \to p_0$	3.2:	$\mathcal{T}(p_1) = 114 + 3 = 117$
	$\mathcal{T}(p_0) = 0 + 75 = 75$	3.3:	$\mathcal{T}(p_0) = 79 + 2 = 81$
3.1:	$z_2 \to p_1$	3.1:	$z_1 \to p_0$
	$\mathcal{T}(p_1) = 0 + 60 = 60$		$\mathcal{T}(p_0) = 81 + 40 = 121$
3.2:	$\mathcal{T}(p_1) = 60 + 4 = 64$	3.2:	$\mathcal{T}(p_0) = 121 + 1 = 122$
3.3:	$\mathcal{T}(p_0) = 75 + 4 = 79$	3.2:	$\mathcal{T}(p_0) = 122 + 1 = 123$
		3.3:	$\mathcal{T}(p_1) = 117 + 3 = 120$
		3.3:	$\mathcal{T}(p_1) = 120 + 2 = 122$

(a) (b)

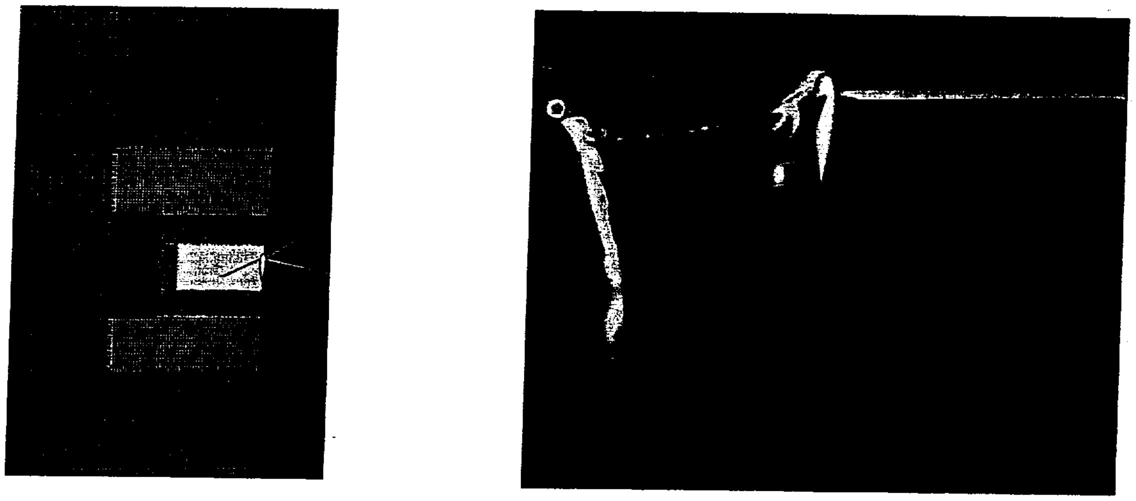
Figure 3: (a) Graph representation of overset grid system in Fig. 1; (b) stepping through the LTF_MFT_ACC procedure in Fig. 2.

in step 3.1. In step 3.2, we need to look at all grids assigned to processors other than p_1 that are also receivers of inter-grid data from z_2. This set of grids is denoted as $R(z_2)$ in Fig. 2. The graph in Fig. 3(a) shows that z_1 and z_3 are in $R(z_2)$; however, z_1 is still unassigned. Since z_3 belongs to p_0, the communication overhead C_{23} is added to $\mathcal{T}(p_1)$; hence, $\mathcal{T}(p_1) = 64$. Similarly, in step 3.3, the set $D(z_2)$ consists of grids that are donors of inter-grid data to z_2. Because z_1 is unassigned and z_3 is mapped to p_0, $\mathcal{T}(p_0)$ is updated with C_{32}; thus, $\mathcal{T}(p_0) = 79$. The remainder of the grouping procedure is shown in Fig. 3(b). Notice that the parallel execution time is 123 ($\max_j \mathcal{T}(p_j)$) whereas the serial execution time is 225 ($\sum_i X_i$).

One important advantage of EVAH over standard graph partitioners like Chaco is that EVAH is able to handle directed graphs. This is critical for overset grid systems where the volume of inter-grid data is not necessarily symmetric between two overlapping grids. For example, $C_{01} \neq C_{10}$ in Fig. 3. For our experiments with Chaco, we generated undirected graphs by collapsing parallel edges and setting the edge weight to be the average of the two weights. Therefore, C_{01} and C_{10} would be replaced by a single undirected edge with weight 1.5.

4 Experimental Results

The CFD problem used for the experiments in this paper is a Navier-Stokes simulation of vortex dynamics in the complex wake flow region around hovering rotors [14]. The Cartesian off-body wake grids surround the curvilinear near-body grids with uniform resolution, but become gradually coarser upon approaching the outer boundary of the computational domain. The overset grid



(a) (b)

Figure 4: (a) A cut through the overset grid system surrounding the hub and rotors; (b) computed vorticity magnitude contours on a cutting plane located 45° behind the rotor blade.

system consists of 857 blocks and approximately 69 million grid points; Fig. 4(a) shows a cut through it. Figure 4(b) presents computed vorticity magnitude contours on a cutting plane located 45° behind the rotor blade. All experiments were run on the 512-processor 400 MHz SGI Origin2000 shared-memory system at NASA Ames Research Center. Due to the size of the test problem, our runs were conducted for only 100 time steps. The timing results are averaged over the number of iterations and given in seconds.

Table 1 shows a comparison of the three load balancing strategies: bin-packing, Chaco, and EVAH, for a range of processor sets. The execution time T_{exec} is the average time required to solve every time step of the application, and includes the computation, communication, Chimera interpolation, and processor idle times. The average computation (T^{avg}_{comp}) and communication (T^{avg}_{comm}) times over P processors are also shown. Finally, the maximum computation (T^{max}_{comp}) and communication (T^{max}_{comm}) times are reported and used to measure the quality of load balancing for each run. The computation load balance factor (LB_{comp}) is the ratio of T^{max}_{comp} to T^{avg}_{comp}, while the communication load balance factor (LB_{comm}) is the ratio of T^{max}_{comm} to T^{avg}_{comm}. Obviously, the closer these factors are to unity, the higher is the quality of load balancing.

Notice that the computational workload for all three strategies is identical as is evidenced by the fact that T^{avg}_{comp} is essentially the same for a given value of P. However, T_{exec} is almost always the lowest when using the EVAH-based task assignment heuristic for load balancing. This is expected since the goal of this strategy is to minimize the total execution time. On the other hand,

Table 1: Runtimes (in seconds) and load imbalance factors with bin-packing, Chaco, and EVAH grid grouping strategies

	Bin-packing algorithm						
P	T_{exec}	T_{comp}^{max}	T_{comp}^{avg}	T_{comm}^{max}	T_{comm}^{avg}	LB_{comp}	LB_{comm}
32	32.0	29.3	23.0	1.30	0.90	1.27	1.44
64	13.5	11.9	10.8	0.67	0.55	1.10	1.22
128	7.6	6.2	5.4	0.60	0.52	1.15	1.15
256	5.5	3.7	2.8	0.88	0.50	1.32	1.76
320	4.7	2.9	2.2	0.57	0.46	1.32	1.24
384	4.7	2.9	1.9	0.99	0.56	1.53	1.77
448	4.5	3.0	1.7	0.85	0.46	1.76	1.85

	Chaco graph partitioning heuristic						
P	T_{exec}	T_{comp}^{max}	T_{comp}^{avg}	T_{comm}^{max}	T_{comm}^{avg}	LB_{comp}	LB_{comm}
32	26.9	23.6	21.0	0.80	0.70	1.12	1.14
64	13.7	12.3	11.0	0.42	0.37	1.12	1.14
128	7.5	6.5	5.4	0.32	0.28	1.20	1.14
256	4.7	3.9	2.7	0.26	0.21	1.44	1.24
320	4.6	3.3	2.2	0.34	0.28	1.50	1.21
384	4.9	3.5	1.9	0.42	0.34	1.84	1.24
448	4.5	3.1	1.7	0.41	0.35	1.82	1.17

	EVAH task assignment heuristic						
P	T_{exec}	T_{comp}^{max}	T_{comp}^{avg}	T_{comm}^{max}	T_{comm}^{avg}	LB_{comp}	LB_{comm}
32	26.2	23.3	21.8	1.50	1.28	1.07	1.17
64	13.0	11.3	10.8	0.76	0.66	1.05	1.15
128	7.3	5.8	5.4	0.99	0.89	1.07	1.11
256	4.8	3.2	2.7	0.77	0.67	1.19	1.15
320	4.4	3.0	2.3	0.76	0.65	1.30	1.17
384	4.7	3.1	1.9	0.97	0.55	1.63	1.76
448	4.3	3.0	1.7	0.73	0.64	1.76	1.14

the communication times T_{comm}^{max} and T_{comm}^{avg} for Chaco are about a factor of two lower than the corresponding timings for the bin-packing algorithm and EVAH. This is because standard graph partitioners like Chaco strive to minimize total communication time.

The overall quality of computational workload balance for the three strategies can be observed by comparing LB_{comp} from Table 1. EVAH returns the best results, demonstrating its overall superiority. Rather surprisingly, Chaco performs worse than bin-packing. We believe that this is because graph partitioners are usually unable to handle small graphs containing only a few hundred nodes. Obviously, LB_{comp} increases with P since load balancing becomes more challenging with a fixed problem size. Results in Table 1 also show that the quality of communication load balancing LB_{comm} is a big drawback of the bin-packing algorithm. Both Chaco and EVAH improve this factor considerably. In fact, except for $P = 384$, LB_{comm} using EVAH is at most 1.17.

It should be noted here that though we are evaluating LB_{comp} from the computational runtimes, the grid grouping strategies are all based on the number of weighted grid points. The three left plots in Fig. 5 show the distribution of weighted grid points across 64, 128, and 256 processors for all three load balancing strategies. The "predicted" values of LB_{comp} are also reported and are typically somewhat better than those computed from actual runtimes (see Table 1), but demonstrate the same overall trend.

The three plots on the right in Fig. 5 present a more detailed report of the execution, computation, and communication times per processor for $P = 64$, 128, and 256. Due to the synchronization of MPI processes, T_{exec} is independent of the processor ID and are shown at the top of the plot area for each case. For the sake of clarity, T_{comm} is shown at a different scale indicated by the right vertical axis. Observe that the EVAH T_{comp} curves are consistently much smoother than those for the other grouping strategies. For the T_{comm} curves, both Chaco and EVAH show a much more uniform distribution across processors than bin-packing.

Scalability beyond 256 processors is generally poor as a result of using a fixed problem size that is independent of the number of processors. For example, when $P = 448$, each group contains, on average, only two grids (since $Z = 857$). With such a low number of blocks per group, the effectiveness of any strategy is diminished; moreover, the communication overhead relative to computation increases substantially. For instance, with low processor counts, the communication-to-computation ratio is less than 6%, but grows to more than 37% with higher counts.

5 Summary and Conclusions

The multi-block overset grid technique is a powerful tool for solving complex realistic computational fluid dynamics (CFD) problems at high fidelity; however, parallel performance depends critically on the quality of load balancing. In this paper, we presented and analyzed three different load balancing strategies for overset grid applications. The first used a simple bin-packing algorithm while the other two were based on graph partitioning techniques.

All experiments were performed with the OVERFLOW-D Navier-Stokes code on a 512-processor Origin2000 system at NASA Ames Research Center. The CFD application was the simulation of

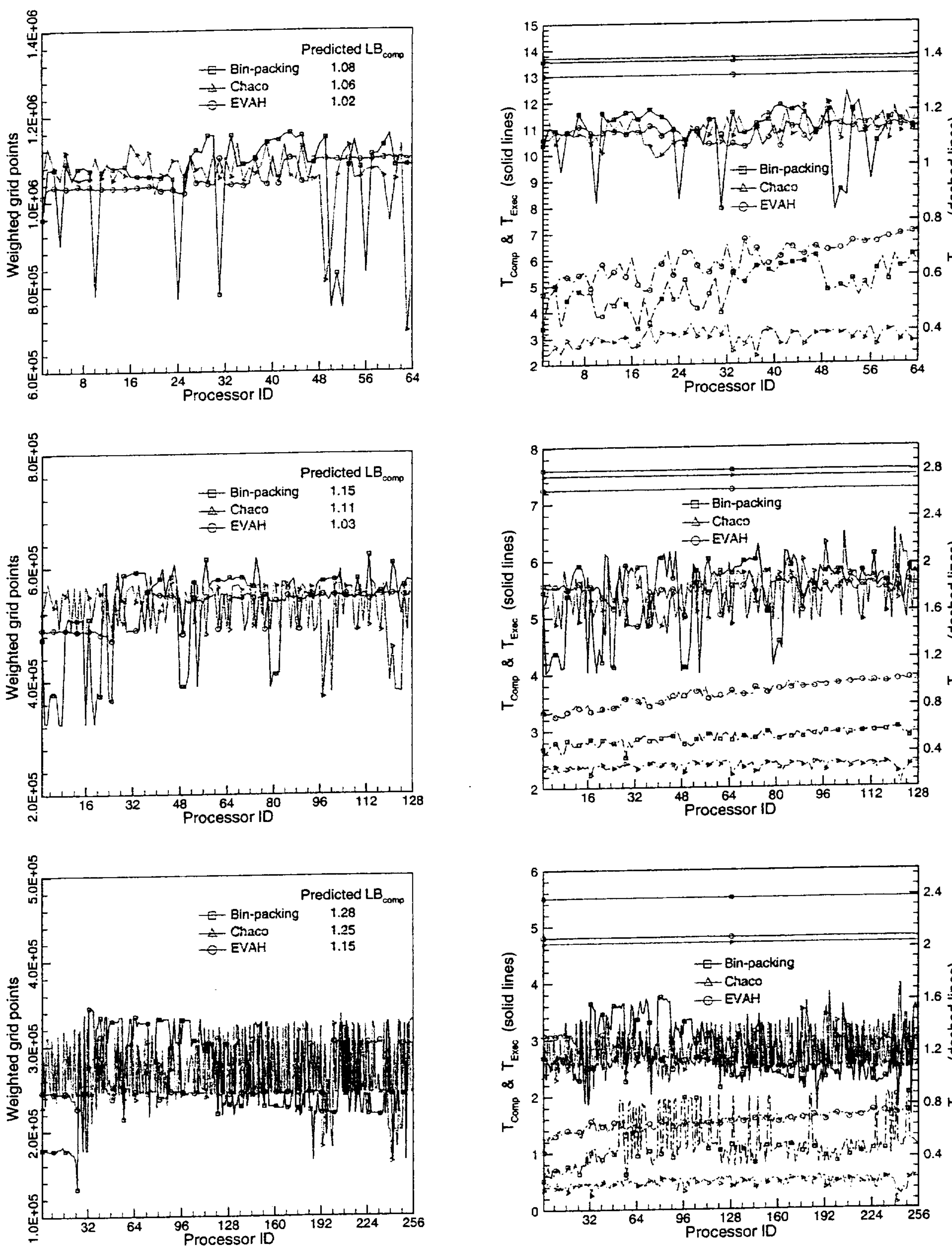


Figure 5: Distribution of weighted grid points (left) and execution, computation, and communication times (right) per processor for $P = 64$, 128, and 256 (top to bottom).

vortex dynamics in the complex flow region around hovering rotors. The grid system consisted of 857 blocks and almost 69 million grid points. Overall results indicated that graph partitioning-based strategies perform better than naive bin-packing due to their more comprehensive consideration of grid connectivity. However, standard graph partitioners did not perform too well because of their inability to effectively handle small graphs. Instead, a grid clustering technique based on task assignment heuristics with the goal of reducing the total execution time performed extremely favorably by improving the communication balance.

Further improvements in the scalability of the overset grid methodology could be sought by using a more sophisticated parallel programming paradigm especially when the number of blocks Z is comparable to the number of processors P, or even when $P > Z$. One potential strategy that can be exploited on SMP clusters is to use a hybrid MPI+OpenMP multilevel programming style [4]. This approach is currently under investigation.

Acknowledgements

The work of the first author was supported by NASA Ames Research Center under Contract Number DTTS59-99-D-00437/A61812D with AMTI/CSC.

References

[1] R. Biswas, S.K. Das, D.J. Harvey, and L. Oliker, "Parallel dynamic load balancing strategies for adaptive irregular applications," *Applied Mathematical Modelling*, 25 (2000) 109–122.

[2] P.G. Buning, W. Chan, K.J. Renze, D.Sondak, I.T. Chiu, J.P. Slotnick, R. Gomez, and D. Jespersen, *Overflow User's Manual Version 1.6au*, NASA Ames Research Center, Moffett Field, CA, 1995.

[3] M.J. Djomehri, R. Biswas, R.F. Van der Wijngaart, and M. Yarrow, "Parallel and distributed computational fluid dynamics: Experimental results and challenges," in: *Proc. 7th Intl. High Performance Computing Conf.*, LNCS 1970 (2000) 183–193.

[4] M.J. Djomehri and H. Jin, "Hybrid MPI+OpenMP programming of an overset CFD solver and performance investigations," NASA Ames Research Center, NAS Technical Report NAS-02-002 (2002).

[5] B. Hendrickson and R. Leland, *The Chaco User's Guide Version 2.0*, Tech. Rep. SAND95-2344, Sandia National Laboratories, Albuquerque, NM, 1995.

[6] B. Hendrickson and R. Leland, "An improved spectral graph partitioning algorithm for mapping parallel omputations," *SIAM J. on Scientific Computing*, 16 (1995) 452–469.

[7] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. on Scientific Computing*, 20 (1998) 359–392.

[8] N. Lopez-Benitez, M.J. Djomehri, and R. Biswas, "Task assignment heuristics for distributed CFD applications," in: *Proc. 30th Intl. Conf. on Parallel Processing Workshops*, (2001) 128–133.

[9] R. Meakin, "On adaptive refinement and overset structured grids," in: *Proc. 13th AIAA Computational Fluid Dynamics Conf.*, Paper 97-1858 (1997).

[10] R. Meakin and A.M. Wissink, "Unsteady aerodynamic simulation of static and moving bodies using scalable computers," in: *Proc. 14th AIAA Computational Fluid Dynamics Conf.*, Paper 99-3302 (1999).

[11] H.D. Simon, "Partitioning of unstructured problems for parallel processing," *Computing Systems in Engineering*, 2 (1991) 135–148.

[12] J. Steger, F. Dougherty, and J. Benek, "A Chimera grid scheme," *ASME FED*, 5 (1983).

[13] R.C. Strawn and J.U. Ahmad, "Computational modeling of hovering rotors and wakes," in: *Proc. 38th AIAA Aerospace Sciences Mtg.*, Paper 2000-0110 (2000).

[14] R.C. Strawn and M.J. Djomehri, "Computational modeling of hovering rotor and wake aerodynamics," in: *Proc. 57th Annual Forum of the American Helicopter Society*, (2001).

[15] R. Van Driessche and D. Roose, "Load balancing computational fluid dynamics calculations on unstructured grids," in: *Parallel Computing in CFD*, AGARD-R-807 (1995) 2.1–2.26.

[16] A.M. Wissink and R. Meakin, "Computational fluid dynamics with adaptive overset grids on parallel and distributed computer platforms," in: *Proc. Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, (1998) 1628–1634.